PLASYS: Final Report

by

Donald Oestreicher

Technical Report 3655

May 1980

Computer Science

California Institute of Technology

Pasadena, California 91125

Silicon Structures Project

sponsored by

Burroughs Corporation, Digital Equipment Corporation,

Hewlett-Packard Company, Honeywell Incorporated,

International Business Machines Corporation,

Intel Corporation, Xerox Corporation,

and the National Science Foundation

PLASYS: Final Report

ABSTRACT

This Silicon Structure Project Report documents an exploratory study of Programmable Logic Array (PLA) optimization. In section I, the report introduces PLAs, and discusses five areas of possible optimization - system design, logic design, circuit design, layout, and fabrication and test. Section II continues with a description of the logic design optimization techniques investigated during this study. Section III is the Users' Guide for the PLA optimization system developed as part of this work. Finally, Section IV presents some conclusions.

The two main conclusions concern the effectiveness of the PLASYS logic optimizations, and the potential applications for PLASYS.

1. Effectiveness - The PLASYS optimizations are not on a par with hand optimization. PLASYS will perform most straight forward optimizations, and remove most trivial coding inefficiencies. However, it can not find the optimal PLA coding.

   It is fair to compare PLASYS with an average present-day compiler. It does not discover all possible optimizations; however, it does make most of the obvious improvements. It performs operations analogous constant term evaluation, and recognition of common sub-expressions and redundant calculations.

2. Application - The primary application for PLASYS is PLAs which can afford to give up some area for other considerations. Other considerations could include:

   - Shortened Design Time - This might be to allow many higher-level design iterations, or to reduce total development time.

   - Higher Level Specification - The need for higher level PLA specification might be to better document the design, or to reduce its complexity, thus increasing the confidence in its correctness.

The report also contains five appendicies. These appendices document the both the tabular and higher-level specifications languages for PLAs. Appendix III gives two examples of PLA design. Appendix IV provides system documentation. Appendix V is a PLA optimization Bibliography.

TABLE OF FIGURES

TABLE OF CONTENTS

Introduction

During the past year I have been looking at the design of Programmable Logic Arrays. This report summarizes that work, and documents the PLA design tools I have developed at Caltech. This report is divided into four sections. Five appendicies are also included to provide more details about PLASYS. The four main report sections are:

I. Programmable Logic Arrays - An Overview
II. PLA Optimization - Introduction to PLASYS
III. PLASYS Users' Guide
IV. PLASYS - Some Preliminary Conclusions

Section I presents an overview of PLAs - briefly covering what they are and how they are used. The major theme is the importance of PLAs (and computer aided PLA design) in a structured design methodology. This section continues with a general discussion of PLA design - introducing PLA logic design as one phase of the PLA design process.

PLASYS emphasizes the logic design phase of PLA optimization. Section II presents the facilities PLASYS provides to optimize PLA logic specifications. Section III is the PLASYS User Guide for Caltech DECSystem-20 users.

Finally, Section IV informally presents the PLASYS experimental results. The major conclusion is that the PLASYS optimization tools are not very powerful, but they can provide assistance in environments where speed and accuracy are the primary concerns. This is true in most university environments, and many industrial applications. The latter include applications where a particular PLA is not in a speed or area critical section of the chip, and applications where large scale complexity problems dominate performance issues.

The five appendicies included with this report are:

I. The Format of PLA Files
II. The Format of PLA Sources
III. Two Examples of PLA Design
IV. PLASYS Implementation and Maintenance
V. PLA Optimization Bibliography

Section I: Programmable Logic Arrays - An Overview

A programmable logic array (PLA) is a generic name used for an array of gates which accepts inputs on one side and produces logical combinations of the inputs on an orthogonal side. Figure I.1 shows a the basic PLA building block.

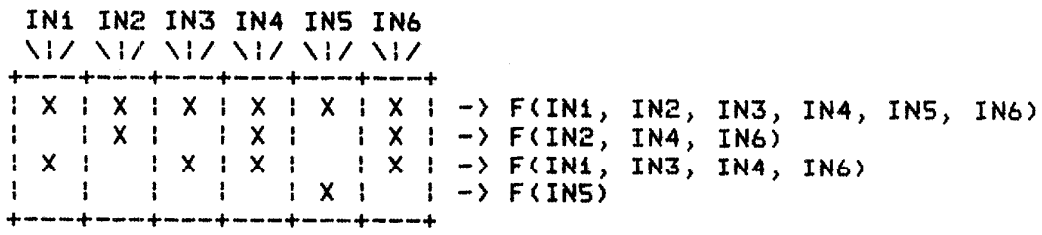


Figure I.1: Basic PLA Building Block

Figure I.1 shows a 6 column by 4 row array. This array contains 24 PLA cells. The cells with the X's are slightly different from those without - usually an added transistor. All cells are the same size - usually square. The function F might be AND, OR, NOR etc.

In most applications two PLA building blocks are connected in series, with the outputs of the first connected to the inputs of the second. In these cases, it is best to think of the first array as implementing the function AND, and the second array the function OR. In fact, both arrays might implement NOR to achieve the same result.

Figure I.2 shows the basic PLA. A PLA has input columns, output columns, and minterms rows.

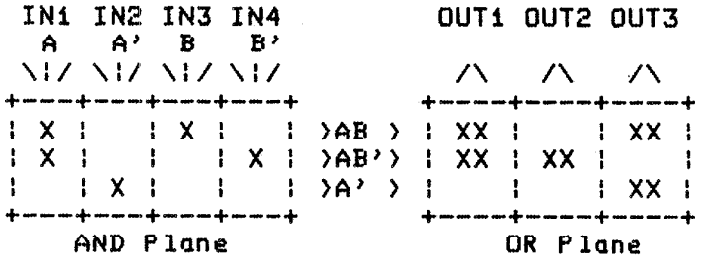


Figure I.2: Basic PLA

In Figure I.2, IN1 might be signal A, and IN2 the complement of A (A'). Similarly, IN3 and IN4 might be B and B' respectively. In this case the three rows of the PLA would represent the products AB, AB', and A'. With these three products to the OR Plane would implement the following three equations:

```
OUT1 = AB + AB'     = A
OUT2 =      AB'     = AB'
OUT3 = AB   +   A'  = A' + B
```

This example demonstrates an interesting feature of PLA designs, even though OUT1=A is "simpler", OUT1=AB+AB' produces a smaller PLA.

In summary, PLAs are regular structures which implement boolean sums of products; any logic equation can be represented as a sum of products. The width of a PLA is basically two units for each input signal (a true column and a complemented column) and one unit for each output. The length is dependent on the number of minterms required.

As a fully general logic element, PLAs can replace block of "random" logic. This includes micro-code controllers, state machines, ALUs etc. PLAs can be designed with or without clocks. Figure I.3 shows a schematic PLA which is clocked and has a state register to implement a finite state machine. In a structured design methodology, the PLA is the escape hatch allowing regular structures to be used for any problem.

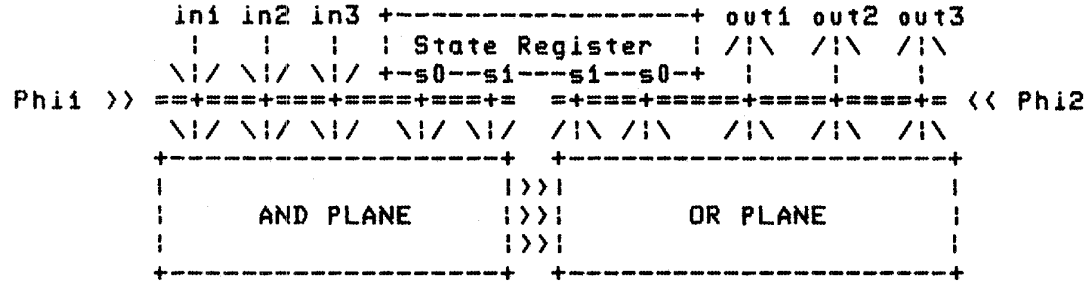


Figure I.3: Clocked PLA State Machine

Beyond simply using PLAs as regular implementations for random logic, many studies have shown PLAs to be useful as units of structured designs. IBM's J.C. Logue suggests that PLAs provide a vehicle to organize the increasing LSI complexity [Log75]. Peter Cook, also of IBM, shows that PLA implementations need not cost more than other approaches [Coo79b]. John Gray of the Caltech Silicon Structure Project suggests that PLAs can be connected in tandem to directly implement RT Level designs. The basic trend show an increasing concern for the complexity of VLSI design, and increased emphasis on PLAs as an important silicon structure. The PLA structure can provide the starting point for a design methodology, and aid the organization of a large design.

Because of the PLA generality, CAD systems targeted to produce PLAs, can accept flexible inputs specifications. The compiler described in Appendix II accepts logic equations and/or state machine descriptions; it automatically generates PLAs. As design correctness becomes more important relative to layout efficiency, the use of automatically

generated PLAs will increase. PLA optimization increases the acceptability of this alternative, by reducing its cost.

In addition to the PLA optimization techniques discussed in this report, other researchers are looking at simple extensions to the basic PLA structure which increase its efficiency. These include an additional level of input decoding [Sch80], and the addition of memory into the PLA structure [Pat79].

The remainder of this section discusses the various issues related to PLA optimization. The earliest and most extensive work in the area was done at IBM [Fle75, Hon74]. Section II continues to present the specific PLA optimization techniques used by PLASYS.

PLA Optimization

PLA design is similar to other IC design functions. It is a five step process. These steps are:

1. System Design
2. Logic Design
3. Circuit Design
4. Layout
5. Fabrication and Test

1. System Design

System design involves structuring, partitioning, and state assignment. Before any logical function is implemented as a PLA, the silicon system containing the PLA must be specified. The chip floor plan determines which signals and functions are localized. It is often counter-productive to run signals long distances when they can be combined where they are generated. This latter approach might lead to more smaller PLAs, or a different functional partitioning between PLAs. In any case, the chip design has strong implication on the individual PLA functional requirements.

As the PLA size is the product of the I/O count and the minterms, two PLAs with independent I/O's often produce a cost savings over a single PLA. These partitioning issues are decided during system design. Some interesting work on hierarchical PLA partitioning was done by Ron Ayres [Ayr79].

When PLAs are used to implement state-machine, the question of state assignment is important. A judicious state varaiable choice can reduce both the PLA's width and its length.

The PLA system design phase produces the functional specifications for all the PLAs used on a chip. These specifications might be represented as logic equations, or state machine descriptions.

Alternatively, the PLA function might be specified tabularly as a set of and/or planes. In any case, the system design goal is to partition the chip's "random" logic requirements into several blocks (PLAs). This top-level design phase also develops the chip floor plan and wiring management strategy.

2. Logic Design

When the system design is complete, each PLA may be considered individually. Following PLA functional design, comes PLA logic design. Each PLA's specification requires it to perform certain logical functions; the logic design phase attempts to minimize the cost of providing these requirements. The main cost considered during logic design is chip area; however, smaller PLAs tend to use less power. In general speed and power performance issues are defered to the next design phase - circuit design.

PLA logic design involves the transformations to the functional specification to take advantage of the PLA structure. In classical logic design each equation is minimized independently, and literal counts are as important as operator counts. In PLAs, all equations in a single block must be minimized together, and the number of literals in a product does not significantly affect the PLA cost.

In a PLA, each product requires one row whether it has one literal (A), or many literals (ABC'DE'G'H'K). Additionally, if terms are used by other equations, it is more efficient to use many terms in place of few (see the OUT1=AB+AB' example above).

PLASYS is concerned with PLA logic design; the PLASYS techniques for logic optimization will be discussed in the next section. The bibliography in Appendix V lists a number of other sources for both PLA logic optimization [Kam79, McC62, Su691, and other PLA design optimizations [Gol80, Wei79, Web79, Woo791.

3. Circuit Design

Circuit design endeavors to optimize the electrical properties of the PLA. During this phase the speed/power/area trade-offs are considered, and the power/ground buses are designed. Circuit analysis is the chief design tool during this phase.

PLA circuit design has two levels of concern. First, the low-level basic PLA cell must meet speed, power, and area requirements. Second, the actual array-level design needs special power, ground, clocks, drivers etc. to interface each particular PLA into its individual chip environment. Some work in this area has been included in the various Caltech PLA implementation sub-systems.

4. Layout

During layout, the PLA is connected to its external signals from the chip. The flexibility of the PLA structure can be extensively exploited at this time to optimize the layout.

PLA layout design is fruitful area for automatic PLA optimization. In addition to having I/O connection at the tops of the planes (see Figure I.2 above), they might equally be placed at the bottoms. If a function consists of only one term, its value might be available at either side of the PLA, thus saving an output column.

Beyond these simple optimizations, it should be noted that the input and output columns (as well as the minterm rows) can be permuted at will. These permutations can be used to optimize the interface with the rest of the chip. They might also be permuted to generate large areas which do not contains any X's. In this latter case, these areas of the PLA need not be included in the layout. If some functions are independent they might share a column - generating one output at the top and another at the bottom. Other functions could share rows - with one minterm going to an or plane to the left and the other minterm to an or plane on the right.

This list of layout optimizations is not complete, but merely suggests some possible PLA optimizations. Some work has been done in industry, but more is possible and needed. Automatic layout optimization can potentially save more area and power than logic and circuit design combined. This is true because layout optimization concerns the reduction to the PLA/chip wiring interface as well as reductions to the PLA itself. In some cases, the wiring interface consumes more space than the arrays.

5. Fabrication and Test

PLA logic gates are different than those available in SSI packages. They are gates which support large fan-in and fan-out factors. PLA structures also provide a great number of places for potential errors which do not even appear in the logic descriptions. Basically, the test stategies developed for SSI logic gates must at least be re-examined before it can be applied to PLAs. On the other side, the PLA structure is highly regular, and should support some automatic testing. Much work is being done in the area of PLA testing [Cha78, Eic80, Ost74, Ost79].

Section II. PLA Optimization - Introduction to PLASYS

Section I described the five areas of PLA design - system, logic, circuit, layout, and test. PLASYS concentrates on logic design. This section discusses various optimization stategies investigated within PLASYS. The six PLA optimization applications within PLASYS can be grouped into three catagories. These catagories are:

A. Single Equation Optimizations,
B. Multiple Equation Optimizations, and
C. ROM Optimizations.

A. Single Equation Optimizatons

A PLA can be represented as a collection of logic equations. Each output column is the equation defined as the sum of the product rows marked. Single equation optimization mainly considers each PLA output column individually. More generally this section considers localized optimizations, while the other optimizations view the PLA system globally.

Single equation optimizations are implemented by three PLASYS application programs - REDUCE, SHRINK and QUINE. REDUCE performs simple logic reduction. SHRINK implements a collection of local optimizations. QUINE does a "Quine-McCluskey" logic minimization [Bre72, Hil68].

REDUCE - REDUCE searches for equations which contain two terms of the form ... and ...*A. ("..." represents any arbitrary product.) The ...*A is unnecessary and removed. Other logic optimizations (e.g. A*B' + B => A + B or A*B' + A*B => A) are not performed because of global considerations; when two minterms are replaced by a single DIFFERENT minterm, the PLA could actually get larger.

For example the system: X=A*B', Y=A*B, Z=A*B'+A*B is better than X=A*B', Y=A*B, Z=A .

SHRINK - Depending on how a PLA is specified, various inefficiencies can or cannot be expressed; others are encouraged. In a tabular format providing access to every transistor, all inefficiencies can be specified. This need not always be the case; for instance, in the Caltech PLA File Format (Appendix I) the designer can not specify terms of the form: A*A'. (This term is always false.)

Conversely, in higher-level specifications (see Appendicies II & III) A*A' terms often appear as the result of abstracting the PLA function. However, PLA compilers rarely generate two PLA rows representing the same minterm - something easily specified in all tabular specification formats.

The SHRINK application is a collection of routines which remove various such inefficiencies from PLAs. These functions are so obvious that it is easy to think of SHRINK as a program which generates a cleaned-up, canonical PLA representation, rather than part of an optimization system. Prehaps these functions should be part of all PLA processing systems as a preliminary (error checking?) step. The SHRINK optimizations are:

1. Unreferenced Terms - Row 1 in the example PLA (Figure II.1 below) shows a PLA with a unreferenced term. The terms A*B in row 1 is not used in the generation of any of the output signals; the unreferenced term routine SHRINK would delete row 1 from this PLA.

2. Empty Terms - Row 2 in the below example shows an empty term. This empty term is being used as part of the sum for equations X = ... and Z = ...; the SHRINK empty term routine would delete row 2.

3. Constant Terms - Row 3 shows a constant term. The equation Z = ... +A*B*B' is not changed when this term is deleted because the product B*B' is always false. Row 3 would be deleted by SHRINK's constant term routine.

4. Duplicate Terms - Rows 4 and 5 both generate the same minterm (A). Row 4 contributes A to the equation X = ...+A, and row 5 contributes A to the equation Z = ...+A. The duplicate term routine would combine rows 4 and 5. The new row is shown as row 8.

5. Unused Inputs - The example does not show any unused inputs. If both the B and B' columns were empty, input B would be unused, and the unused input routine would removed it from the PLA specification. An unused input is one which does not appear in any product (either true or complemented). The astute reader will notice that the input A' is never used; PLASYS considers this optimization to be a layout optimization, not a logic optimization. (This decision is basically technical, as the internal representation can not represent "half" an input column. In addition this optimization depends on the actual PLA implementation system; many implementation schemes do not implement half a column.)

6. Unused Outputs - The output signal Y does not have any terms. The unused output routine will remove it from the PLA specification. This routine also removes output signals which depend on all the product terms; this is assumed to be an error condition.

In the case of feedback terms if either the input or output is unused, the entire term should be deleted (both input columns and output column). SHRINK ignores this issue by never deleting feedback columns. . Some of the SHRINK "optimizations" are more likely to be the result of specification errors, rather than inefficiencies. As with all PLASYS applications, SHRINK reports PLA changes to the designer - the goal is to prevent accidental (automatic) modification of the design intent during optimization. A further note on unused columns: many real PLAs have unused columns and rows for later patches - these can be inserted after logic design (during layout design).

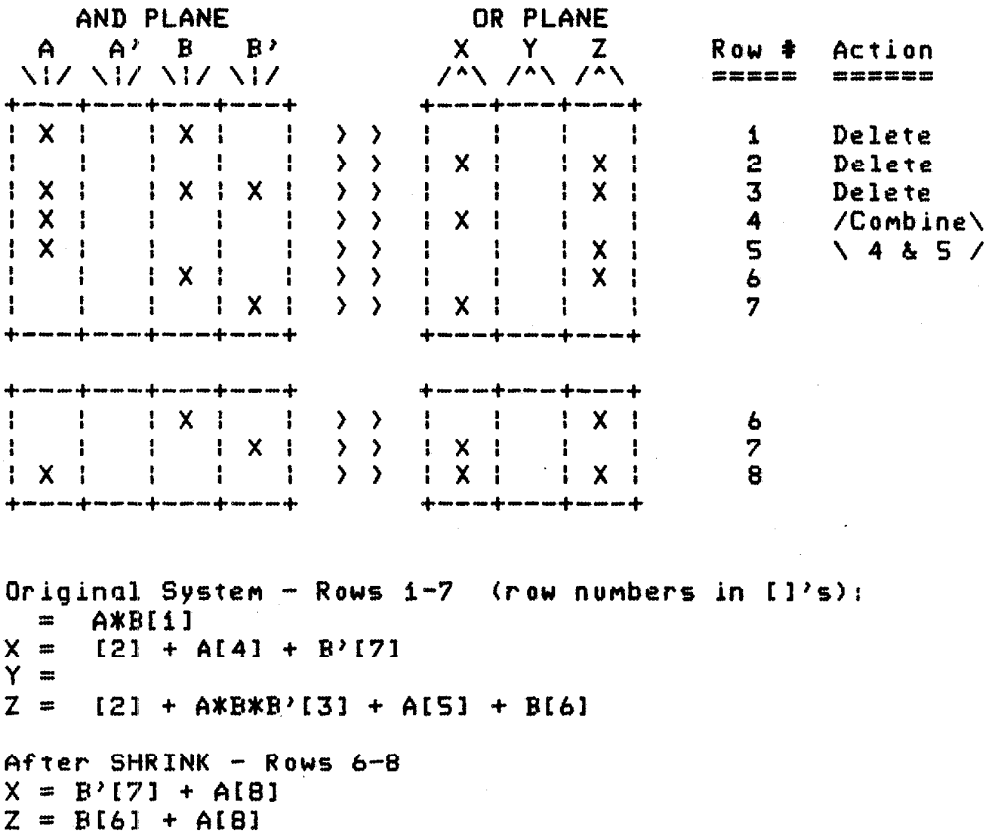


Figure II.1: Example PLA Showing Various Inefficiencies

QUINE - QUINE considers each equation individually and performs a single-output "Quine-McCluskey" logic minimization under the assumption that the other equations will remain unchanged. This implies that any minterms which are used by other equations, will not be deleted from the PLA - their rows can not be reclaimed.

This means that before each equation is processed, each minterm (product) is examined to determined whether it is "shared" with other equations in the PLA, or whether it is "exclusive" to this equation. Shared minterms have several columns marked in the or plane, while exclusive minterms have only a single column marked in the or plane. If a shared minterm can be removed, the PLA size will remain unchanged; a single mark will be removed from the or plane. If an exclusive minterm is removed, the PLA becomes smaller; the entire row is removed, as the or plane has no marks in that row.

QUINE takes advantage of this in two ways. First, if all the minterms are shared, no logic optimization is attempted. Second, all shared minterms are considered to be free; they are represented as don't cares. The standard Quine-McCluskey algorithm prefers to use don't care terms; as a result this approach causes shared terms to be prefered over exclusive ones.

A second version of QUINE (XQUINE) does not take shared minterms into considerations. It simply reduces each equation independently of the PLA system. This result produces equations which are easy to read; the PLA tends to be much larger as sharing tends to be minimized.

B. Multiple Equation Optimizations

In contrast to single equation optimizations, the multiple equation optimizations take a global view of the PLA system. They consider which equation sets employ a particular minterm. Basically where the local optimizations in the previous section considered or-plane columns, these global optimizations consider or-plane rows. An or-plane row represents the equation set employing the minterm defined by the corresponding and-plane row.

The multiple equation optimizations are performed by two PLASYS applications. These are PAIRS and SPREAD. PAIRS performs single Quine-McCluskey steps to combine small terms (many literals) into larger terms (fewer literals). SPREAD performs the opposite transformation. The examples shown in figure II.2 demonstrate how both transformations can improve the PLA design.

PAIRS - The basic PAIRS algorithm looks for pairs of terms of the form ...*A and ...*A'. These products can be replaced by a single product which doesn't include the literal A. This is most advantageous when the same set of equations use both products. In this case, all the equations are reduced together, and the resulting PLA has one fewer row. This is shown in the example below - rows 1a and 1b are optimized to become row 4.

```
X= AB+AB'+...   =>      X= A+...
Y= AB+AB'+...   =>      Y= A+...
```

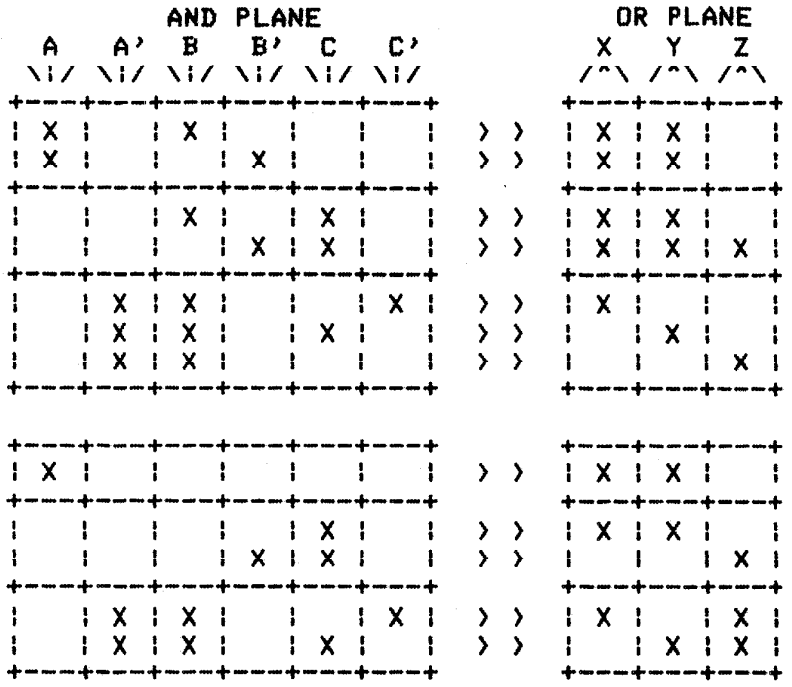
If both terms are not used by the same equations, but rather one is used by a subset of the other (see rows 2a,2b below), then the equations can be simplified, but the PLA remains the same size. The result in this case is shown in rows 5a,5b.

```
X= CB+CB'+...   =>      X= C+...
Y= CB+CB'+...   =>      Y= C+...
Z=    CB'+...   =>      Z= CB'+...
```

The PAIRS application performs this optimization over the entire PLA once. Under some conditions, PAIRS will find additional improvements when executed for a second time.

SPREAD - SPREAD also searches for ...*A and ...*A' pairs. Each time such a compatible pair is located, SPREAD goes on to look for the combined term ...; if such a minterm is present, all equations which depend on ... are redefined to be the sum of ...*A and ...*A'. This allows the row with ... to be removed. The example shows this where rows 3a-3c are replaced by 6a and 6b.

```
X= A'BC'+...     =>     X= A'BC'+...
Y= A'BC+...      =>     Y= A'BC+...
Z= A'B+...       =>     Z= A'BC+A'BC'+...
```



Original System - Rows 1-3 (row numbers in []'s):
X = A*B[1a] + A*B'[1b] + B*C[2a] + B*C'[2b] + A'*B*C'[3a]
Y = A*B[1a] + A*B'[1b] + B*C[2a] + B*C'[2b] + A'*B*C[3b]
Z = B*C'[2b] + A'B[3c]

After PAIRS and SPREAD - Rows 4-6
X = A[4] + C[5a] + A'*B*C'[6a]
Y = A[4] + C[5a] + A'*B*C[6b]
Z = B*C'[5b] + A'*B*C'[6a] + A'*B*C[6b]

Figure II.2: Example PLA Showing Multiple Equation Optimizations

C. ROM Optimizations

In the case of a ROM, the minterms represented in the AND array are mutually exclusive. These terms represent the ROM addresses actually used; no other addresses are expected. MUTEX attempts to modify the AND plane with don't cares, while still maintaining this mutual exclusion property. This reduces the number of transistors in the AND plane.

It was also hoped that loosening the specification in this way might allow some flexibility for the other optimization algorithms. This did not seem to happen.

Section III: PLASYS Users' Guide

The PLASYS Users' Guide is supplied for Caltech users and assumes a working knowledge of the DECSystem-20. This documentation assumes that PLASYS is available on-line on the disk area PS:<SSPLIB.PLA>. The Users' Guide is divided into four sections. These sections are:

A. Getting Started with PLASYS
B. Using PLASYS
C. PLASYS Application Catalog
D. Cleanup after PLASYS

A. Getting Started with PLASYS

In order to be able to use PLASYS, PS:<SSPLIB.PLA> must be on your search list, or the PLASYS message files must be on your disk area. The latter approach is recommended. The PLASYS message files can be created on your disk area by typing "TAKE <SSPLIB.PLA>GETPLASYS" to the executive (the @ prompt). This will copy about a dozen files with the names xxxxxx.MSG into you disk area. These files use very few disk pages.

Once your disk has been initialized in this way, you may execute any PLASYS program by typing "<SSPLIB.PLA>APPLIC" to the executive (where APPLIC is the name of the PLASYS application. All applications create the files PLASYS.CMD and APPLIC.LOG on your disk area.

With your initialized disk, you are ready to use PLASYS. Before anything else you must create a PLA specification. This is done using your favorite text editor in either the high-level PLA language, or tabular form. The high-level specifications are called PLA Sources; they have .SRC extensions, and are described in Appendix II. The tabular specifications are called PLA Files; they have .COD extensions, and are described in Appendix I. All PLASYS applications generate .COD files; .COD files are the standard PLA specification format for Caltech design systems. Plans call for both Simula and ICL design tools to use the standard .COD file format.

B. Using PLASYS

Each PLASYS program is called a PLASYS application. They are all executed in the same way. When they are run they request a PLA name. The PLA name must be 6 characters or fewer. In general (for the PLA named PLANAM), the application will read the PLA File PLANAM.COD and overwrite it with a new version containing the optimized PLA.

As several applications are usually run to process any PLA design, each application writes the last PLA name into the file PLASYS.CMD. If the PLA name request is answered with a carriage return, the application will retreive the name from the PLASYS.CMD file.

This saved PLA name feature simplifies the task of executing a series of PLASYS applications. The applications merely need to be listed in a command file; the "TAKE" of the command file will require the PLA name to be entered once; after that each succeeding step only needs a carriage return. <SSPLIB.PLA>OPTIMIZE is an example of a sequence which optimizes a PLA design and generates the listing file for the newly generated PLA.

As each application runs, its actions are printed on the terminal; these messages are also entered into a log file. The file has the same name as the application, with the extension .LOG (e.g. APPLIC.LOG).

PLA Logic Design Steps - The logic design phase (see Section I for a discussion of PLA design phases) of a PLA design using PLASYS usually goes through three steps. These steps are:

1. Logic Design Specification Step
2. Logic Design Optimization Step
3. Logic Design Verification Step

1. Logic Design Specification Step

The specification step differs with specification approach. If the PLA specification is tabular then a .COD file is generated directly. If the PLA is generated using the high-level PLA Language, a .SRC file is generated and encoded into the tabular format by the PLASYS application ENCODE.

Tabular PLA Specification - In the case of tabular specification the .COD file is generated directly. PLASYS supports several .COD file format extensions to make this approach easier. These are all documented in Appendix I. In addition the program <SSPLIB.PLA>ROTATE reads text files and "rotates" them. This means that the first character of each line becomes the first line of the new output file, the second characters become the second line, etc. In certain applications this use useful during the process of creating tabular specifications.

High-level PLA Specification - Appendix II describes the PLA Language supported by PLASYS. An example of this approach to PLA design is shown in Appendix III. Most applications

expect .COD files; they will process any tabular specification (Standard .COD Format or Extended .COD Format). One application accepts PLA Source Files. This is ENCODE. ENCODE compiles PLA Sources (.SRC files) into PLA Files (.COD Files). Except for having the input extension of .SRC, ENCODE runs like the other applications described above.

2. Logic Design Optimization Step

The optimization step assume the existence of a PLA File (.COD extension) created during the specification step (described directly above). In PLASYS, optimization consists of executing various PLASYS applications; each application will either report no progress and NOT create a new version of the PLA File, or report improvement and create a new version.

The sequence of applications, and their chances of success depends on a number of factors (see Sections II and IV). The optimization applications are described above in Section II: PLA Optimization - Introduction to PLASYS. They are briefly documented below in sub-section C (PLASYS Application Catalog) of this Users' Guide. The following is a listing of <SSPLIB.PLA>OPTIMIZE and represents a reasonable optimization sequence:

```
<SSPLIB.PLA>SHRINK
<SSPLIB.PLA>QUINE
<SSPLIB.PLA>SPREAD
<SSPLIB.PLA>REDUCE
<SSPLIB.PLA>PAIRS
<SSPLIB.PLA>SHRINK
<SSPLIB.PLA>PPRINT
```

3. Logic Design Verification Step

PLASYS provides two tools for verification. The first is a "pretty printer" - PPRINT. In addition to the standard request for PLA File name, PPRINT also asks for a print code. The codes control how the PLA File is printed. The resulting printout of the PLA File is always PLANAM.LST, where PLANAM is the name of the PLA File.

There are four sections to a PLA File printout. They are statistics, block format, normal equations, and checkout equations. Statistics report on PLA size and percent zeros, ones etc. Block format shows a stylized printout of the PLA - and-plane, or-plane, inputs and outputs labels at the top, and a row for each minterm. Normal equations are just

regular logic equations. Checkout equations show the minterms compacted to aid manual checkout.

All printouts include statistics and block format. Print mode S (for short) includes nothing else. E (for equations) adds the normal equations. C (for checkout) adds the checkout equations. L (for long) includes everything. ? <carriage return> prints some documentations and requests the print code again. <carriage return> alone is the same as S.

The second verification tool is the "optimization" application XQUINE. This application performs a Quine-McCluskey type of optimization on each equation independently. In general this makes the PLA larger, but the resulting equations are easier to understand. It is also reasonable to expect all versions of a small PLA to generate the same result from XQUINE.

C. PLASYS Application Catalog

A full PLA design example is given in Appendix III below. Examples for many of the individual applications are given in Section II above.

1. Logic Design Specification Step

CHECK - input: PLANAM.COD - output: PLANAM.COD

Check does not substantially change the PLA File. However, it always writes a new PLA File. Thus, any random manually-generated PLA File can be converted into a sorted standard PLA File by using CHECK.

ENCODE - input: PLANAM.SRC - output: PLANAM.COD

This is the application which compiles PLA Sources into PLA Files. Most PLA design tools requires a PLA File (.COD extension). This single application is the link between the high-level PLA specification langauge (Appendix II) and the rest of the Caltech PLA design tools.

ROTATE - input: TEXT.IN - output: TEXT.OUT

ROTATE reads a text files and creates a new text file. The first line of the new file contains all the first characters of the lines of the old file. Similarly, the second line of the new contains all the second charcters from the old, and so on. This is sometimes useful when generating tabular PLA specifications from data organized by column not row.

2. Logic Design Optimization Step

MUTEX - input: PLANAM.COD - optional output: PLANAM.COD

MUTEX assumes that all minterms are mutually exclusive (as in the case of "ROM" address decoding). Taking advantage of this assumption, MUTEX attempts to add don't cares to the PLA specification. If nothing else, this reduces the number of transistors in the eventual design.

PAIRS - input: PLANAM.COD - optional output: PLANAM.COD

PAIRS searches the PLA specification for compatible minterms. These are minterms which represent identical products, except for a single literal which is true in one and complemented in the other. A*B'*C' and A*B*C' are compatible. If the equations using one minterm are a subset of the equations using the other, the specification can be simplified. If both minterms are used by the same equations, they are be replaced by a single combined minterm.

REDUCE - input: PLANAM.COD - optional output: PLANAM.COD

REDUCE does logic reduction on single equations independently. As such an approach can often be counter-productive (see XQUINE), only the "safe" reduction: A+AB+.... => A+.... is implemented.

SHRINK - input: PLANAM.COD - optional output: PLANAM.COD

SHRINK makes a number of straight forward "cleanup" optimizations on the PLA File. These optimizations are: 1. remove unused input columns and unused output columns, 2. remove minterm rows which have errors, are not referenced by output columns, or do not reference any inputs, and 3. merge duplicate minterm rows.

SPREAD - input: PLANAM.COD - optional output: PLANAM.COD

SPREAD also (see PAIRS above) looks for compatible minterms. As each pair of compatible minterms is located, the combined minterm is generated (e.g. A.C' for A.B.C' or A.B'.C') and searched for in the PLA specification. If the combined minterm is located, its equations are spread over the compatabile minterms, and it is removed.

QUINE - input: PLANAM.COD - optional output: PLANAM.COD

QUINE does a modified Quine-McCluskey type of logic minimization on the PLA File. This minimization takes the PLA multiple-outputs into account, while still working on a single equation at a time. Section II has more details.

3. Logic Design Verification Step

PPRINT - input: PLANAM.COD - output: PLANAM.LST

PPRINT makes a pretty printout of the contents of a PLA File. It has several options discussed above. When PPRINT asks for print code, a ? <carriage return> will prompt the various options and their associated codes.

XQUINE - input: PLANAM.COD - output: PLANAM.COD

XQUINE does a single-output Quine-McCluskey type logic minimization on each logic equation independently of the others. This usually results in a larger, less efficient PLA specification. However, the logic equations are easier to read and verify for correctness. It is a good practice to make a temporary copy of the PLA File before using XQUINE; XQUINE often pessimizes the implementation.

D. Cleanup after PLASYS

Following the use of PLASYS, only the PLA Files (.COD) and Sources (.SRC) are useful. However, a casual examination of the disk will reveal many other files. Most of these can be removed with a "TAKE <SSPLIB.PLA>DELPLASYS". This command file deletes all PLASYS message files (.MSG), log files (.LOG), and the saved PLA name file (PLASYS.CMD). Users might also wish to delete *.LST. If ROTATE was employed *.IN and *.OUT may also need to be deleted.

Section IV: PLASYS - Some Preliminary Conclusions

These conclusions are based on a small number of experiments with some Caltech designs, and some small industrial designs. The general observation that PLASYS optimizations are not as complete as hand optimizations is balanced against the growing number of PLA applications where minimum speed and area are not the primary concerns. PLASYS will perform most straight forward optimizations, and remove most trivial coding inefficiencies. However, it can not find the optimal PLA coding.

It is fair to compare PLASYS with the average present-day compiler. It does not discover all possible optimizations; however, it does make most of the obvious improvements. It performs operations analogous constant term evaluation, and recognition of common sub-expressions and redundant calculations. Constant term evaluation provides a limited "conditional assembly" capability.

Positive results can be reported in three areas:

A. Applications for PLASYS
B. Results with Specific PLASYS Algorithms
C. Directions for Further Work

A. Applications for PLASYS

The small number of examples showed that some industrial designs could be improved by PLASYS. In all these cases, it was clear that the logic designer had little concern for the PLA's size. Either the PLA's size was not a limiting factor in chip's floor plan and/or the PLA's correctness or simplicity was more important. Other factors in favor of unoptimized designs are short design times, and later plans to modify the design.

In all cases, a basic situtation involved a logic designer using a tabular PLA specification, and NO optimization tools. The introduction of a higher-level specification language, and simple PLASYS optimizations aids could easily produce 10% improvements. Beyond this improvement, the logic designer would be in a position to accept and use better optimization tools as they become available.

Beyond low priority designs, PLASYS is also a useful aid during system design. Though PLASYS does not give the ultimate minimization, it gives a quick approximation to the final size. Thus, during system design, where many different problem formulations might be tried, PLASYS could automatically generate logic implementations for reasonable comparisons. Two designs could be more fairly compared if they had first been optimized by PLASYS.

B. Results for Specific PLASYS Algorithms

The value of the different PLASYS applications differed from design to design; however, a trend was evident. Each application is discussed individually below.

REDUCE - single equation logic reduction - This application never was helpful. In addition, the PLA File Format is not conducive to this type of optimization. Therefore, not only is this application of little value, it is very expensive to run.

SHRINK - simple single equation optimizations - This application almost always finds some simplifications. This is basically a cleanup operation. Some of the "optimizations" should never happen - they actually represent various PLA specification errors, rather than optimizations. So SHRINK is also useful to check the PLA specification.

QUINE - single output Quine-McCluskey - QUINE usually simplified the equations without reducing the PLA's size. This saved some by reducing the number of transistors. QUINE was able to simplify the equations because tabular specifications encourage efficient, but redundant specifications. Often QUINEs main effect was to make the equations more complex.

On the other hand, XQUINE simplifies all the equations at the cost of increasing the PLA size. This is a useful design checking tool.

PAIRS - combine compatible terms - This application runs very quickly using the PLA File Format with some internal pre-processing to facilitating quick compatible term checking. This application produced almost all the PLA size reductions, and reduced all PLAs which were improved. It seems to find most the cases found by QUINE, at a much smaller CPU and memory cost.

SPREAD - distributing terms over smaller ones - This application which is the dual of PAIRS never helped. Several considerations might have caused SPREAD to fail. While, PAIRS can create the new minterms it needs to optimize the PLA, SPREAD expects them to be already represented from some other source. Thus, if several large terms might be spread over fewer smaller ones, SPREAD would not discover this unless ALL the smaller ones were present. Prehaps a better spread algorithm would be more successful here.

For the same reasons SPREAD requires the different parts of the design to be done at different levels of detail (creating terms of different sizes). This does not often seem to be the case; most design styles seem to produce uniform levels of specification across the entire PLA specification.

MUTEX - ROM optimization - This application did not seems to solve the problem with ROMs - they can not be optimized because they are so tightly specified.

C. Directions for Further Work

Logic design can use further work in three areas. These areas are:

1. Specification
2. Quick Optimization
3. ROM Design

1. Specification

As more design is done with PLAs, higher level specification vehicles are going to be needed. This need is similar to the forces for higher level languages for software. The future important PLA issues will be design time and correctness.

As chips become larger and more complex, the number and size of PLAs will continue to grow. This means more PLAs will NOT be in critical sections of the chip. They will perform many logical functions which are neither time nor space critical. However, their design cost must be kept down. This will require ways to quickly design PLAs and quickly verify their correctness. These goals can not be met by tabular specification tools.

The PLA specification language presented here is but a start; ultimately each PLA application area (instruction decode, I/O controller, random logic etc.) will require a special language to optimize the designers' time.

Some of the problems which were addressed in the PLA specification language in PLASYS include:

- Horizontally Coded Fields - In many PLA applications the input and/or output signals form fields. These fields might be interpretted as integers, characters, arrays of sub-fields etc. The # construct in the PLA Source language is just a start at supporting this need.

- Finite State Machines - When PLAs are used to implement finite state machines, the specification is considered in a fundamentally different way from when it is used to implement logic blocks. The feedback construct in PLASYS addresses the minimal support in this area.

- o Levels of Abstraction - Even though the PLA is finally coded at the transistor on/off level, the designer will want to create higher level abstractions (conditional, with parameters). The text substitution facility (SHORT FOR) is a attempt to meet some of these needs.

With good PLA specification languages the transistion from system design to circuit design should be practically automatic (in the case of non-critical PLAs).

2. Quick Optimizations

Again in the area of non-critical PLAs, inexpensive (in real-time and computer resources) optimization tools allow iteration at the system design level. This starts to attack the important system design problems. System design considers structural alternatives. A system which can consider structural implications in detail (optimized PLAs provide a better basis for comparison than equations) develops better system designs.

3. ROM Design

PLAs are often used to implement sparse ROMs. For some reason these PLAs do not optimize with the same algorithms as other (control) PLAs. Tools to improve ROM implementations would be very useful.

Appendix I: The Format for PLA Files

A PLA is stored on the disk as a .COD file. The standard format for this file is described in: SSP File #3523 -- Proposed .COD File Formats by Dave Johannsen. All PLASYS programs accept a format which includes this standard. This format is called the PLASYS .COD File Input Format; it is documented below. All PLASYS programs generate a more restricted format than the standard. This format is called the PLASYS .COD File Output Format -- also described below.

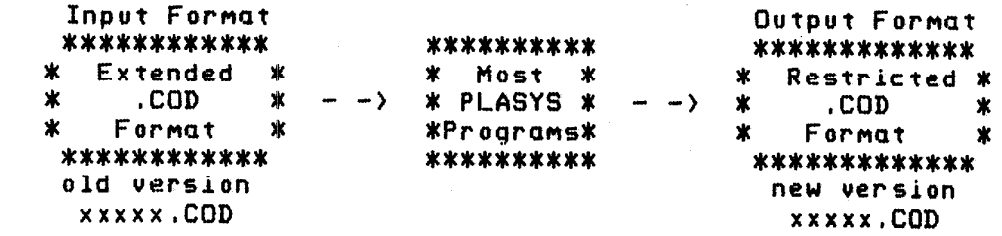


Figure AI.1: Basic PLASYS Information Flow

The PLASYS CHECK program does little more than convert Input Format files to Output Format files (It also sorts the and-plane.) If a PLA application can require all its users to preprocess their .COD files with CHECK, that application can simplify its input parser by only accepting PLASYS .COD File Output Format. This would simplify the application, while at the same time, allowing users to prepare .COD files using the more general PLASYS .COD File Input Format.

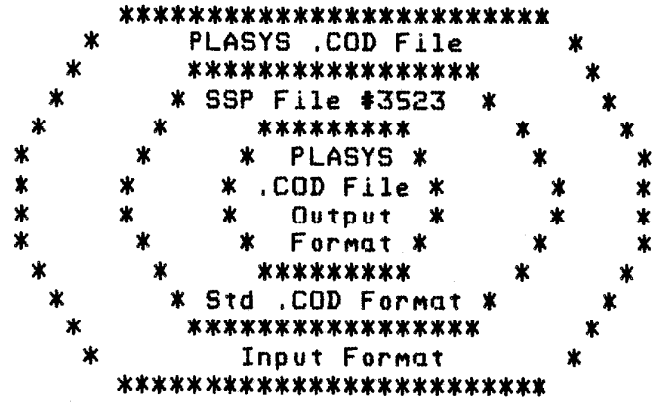


Figure AI.2: Tabular PLA Specification Formats (.COD Files)

A PLA description has four components (see Figure AI.3 below). These are the and-plane, the or- plane, the signal names, and the documentation. The .COD file contains documentation in two places. The header line contains a title and some parameters (count of inputs, outputs, feedbacks, and minterms). Following the plane descriptions and

signal names, the .COD file contains free-format comments. PLASYS programs maintain and append to the first block of comments.

The plane descriptions contain one line for each minterm. These lines start with an and-term descriptor for each and-plane column; the total number of columns is the the inputs plus the feedbacks. The and-plane description is followed by the or-plane description -- an or-term descriptor for each output column (feedbacks + outputs). Finally, the plane descriptions are followed by the input and output signal names (one per line).

```
Inputs=1 Feedbacks=2 Outputs=2 Minterms=5  Sample PLA
000 0i0ix
010 1i10x
1i1 1ixxx
111 00xx1
1x0 xx111
Input-Alpha
A.Result
B.Result

Comment Block
Free Format Comments
```

Figure AI.3: Sample .COD File

The specific details for each PLASYS .COD File Format are found below. SSP File #3523 is the standard and should be read along with these descriptions. The standard is intended to be more stable than any of the PLASYS formats. Dave's document also contains information on PLA semantics and implementations.

The remainder of this appendix contains formal descriptions of the following:

A. PLASYS .COD File Output Format
B. PLASYS .COD File Input Format
C. Extensions of PLASYS .COD File Input Format

Note: ()'s enclose names of non-terminals. ()'s enclose sub-expressions. "'s enclose literal strings. /'s enclose descriptions not further expanded (e.g. /decimal integer/). []'s enclose optional parts of the syntax (e.g. [(a)] is the same as (a) | /nothing/. (xyz)(n) means that xyz must appear exactly n times.

Syntax of .COD Files Generated by PLASYS Applications

PLASYS .COD FILE OUTPUT FORMAT

```
.COD file          := ⟨dimension line⟩ ⟨body⟩ ⟨name block⟩
                      ⟨comment lines⟩
⟨dimension line⟩   := "I=" ⟨inputs⟩ ⟨spc⟩ "F=" ⟨feedbacks⟩ ⟨spc⟩
                      "O=" ⟨outputs⟩ ⟨spc⟩ "M=" ⟨minterms⟩ ⟨spc⟩
                      ⟨title⟩ ⟨eol⟩
 ⟨inputs⟩          := /decimal integer/
 ⟨feedbacks⟩       := /decimal integer/
 ⟨outputs⟩         := /decimal integer/
 ⟨minterms⟩        := /decimal integer/
 ⟨title⟩           := /any string not including ⟨eol⟩/
⟨body⟩             := { ⟨code line⟩ } (minterms)
⟨code line⟩        := { ⟨bit⟩ } (inputs) { ⟨bit⟩ } (feedbacks) ⟨spc⟩
                      { ⟨bit⟩ } (feedbacks) { ⟨bit⟩ } (outputs) ⟨eol⟩
⟨bit⟩              := ⟨true⟩ ! ⟨false⟩ ! ⟨don't care⟩ ! ⟨error⟩
 ⟨true⟩            := "1"
 ⟨false⟩           := "0"
 ⟨don't care⟩      := "."
 ⟨error⟩           := "?"
⟨name block⟩       := { ⟨name⟩ ⟨eol⟩ } (inputs)
                      { ⟨name⟩ ⟨eol⟩ } (outputs)
 ⟨name⟩            := /any string not including ⟨eol⟩/
⟨comment lines⟩    := ⟨blank line⟩ ⟨comment block⟩ ⟨blank line⟩
 ⟨comment block⟩   := /any number of non-blank lines/
 ⟨blank line⟩      := ⟨eol⟩
⟨spc⟩              := " "
⟨eol⟩              := /end of line/
```

SAMPLE OUTPUT FORMAT .COD FILE

```
I=2 F=3 O=1 M=3 Sample Output Format .COD File
00011 1011
01101 1100
11110 0111
Primary Input
Secondary Input
Resulting Output

COMMENTS ...
```

Syntax of .COD Files Accepted by PLASYS Applications
PLASYS .COD FILE INPUT FORMAT

```
.COD file          := <dimension line> <body> [ <name block> ]
                             [ <comment lines> ]
<dimension line>   := ( <dimension> <spc> ) (4) [ <title> ] <eol>
<dimension>        := <i spec> ! <f spec> ! <o spec> ! <m spec>
 <i spec>          := ( "I" ! "i" ) <filler> "=" <inputs>
 <f spec>          := ( "F" ! "f" ) <filler> "=" <feedbacks>
 <o spec>          := ( "O" ! "o" ) <filler> "=" <outputs>
 <m spec>          := ( "M" ! "m" ) <filler> "=" <minterms>
  <filler>         := /any string of alphabetics and spaces/
  <inputs>         := /decimal integer/
  <feedbacks>      := /decimal integer/
  <outputs>        := /decimal integer/
  <minterms>       := /decimal integer/
 <title>           := /any string not including <eol>/
<body>             := ( <code line> ) (minterms)
<code line>        := ( <bit> ) (inputs) ( <bit> ) (feedbacks) <spc>
                   ( <bit> ) (feedbacks) ( <bit> ) (outputs)
                   [ <spc> /any string not including <eol>/ ] <eol>
<bit>              := <true> ! <false> ! <don't care> ! <error>
 <true>            := "1" ! "I" ! "i"
 <false>           := "0" ! "O" ! "o"
 <don't care>      := "." ! "X" ! "x"
 <error>           := "?"
<name block>       := ( <name> <eol> ) (inputs)
                   ( <name> <eol> ) (outputs)
 <name>            := /any string not including <eol>/
<comment lines>    := <blank line> <comment block>
                   <blank line> /anything/
 <comment block>   := /any number of non-blank lines/
 <blank line>      := <eol>
<spc>              := /any number of spaces and tabs/
<eol>              := /end of line/
```

SAMPLE INPUT FORMAT .COD FILE
====== ===== ====== ==== ====

```
MTERMS = 3 Ins=2 f=3 Outs=1  Sample Input Format .COD File
oo011 101I
oi101 1100
ii110 011I
Primary Input
Secondary Input
Resulting Output

COMMENTS ...
```

Extensions to Syntax of .COD Files Accepted by PLASYS Applications

EXTENSIONS TO PLASYS .COD FILE INPUT FORMAT

All PLASYS applications accept extended .COD File Input Format. The following changes differentiate an "extended" file from a "normal" file:

```
Extended .COD File := "EXTENDED" <eol> /.COD File/
.....
<code line>      := <and spec> [ <spc> ] <or spec>
                  [ <spc> /any string not including <eol>/ ] <eol>
 <and spec>      := { <bit> } <inputs> { <bit> } <feedbacks>
 <or spec>       := { <bit> } <feedbacks> { <bit> } <outputs>
<bit>            := [ <bit filler> ] (<single bit> ! <triple bit>)
 <bit filler>    := /any number of spaces and commas/
 <single bit>    := <true> ! <false> ! <don't care> ! <error>
  <true>         := "T" ! "t" ! "I" ! "i"
  <false>        := "F" ! "f" ! "0" ! "o"
  <don't care>   := "D" ! "d" ! "X" ! "x" ! "."
  <error>        := "?"
 <triple bit>    := /any digit 0 - 8/
 "fff"  :=       "0"
 "fft"  :=       "1"
 "ftf"  :=       "2"
 "ftt"  :=       "3"
 "tff"  :=       "4"
 "tft"  :=       "5"
 "ttf"  :=       "6"
 "ttt"  :=       "7"
 "ddd"  :=       "8"
```

SAMPLE EXTENDED INPUT FORMAT .COD FILE

```
MTERMS = 3 Ins=2 f=3 Outs=1  Sample Input Format .COD File
ff3 5T
ft5 6F
tt6 3T
Primary Input
Secondary Input
Resulting Output

COMMENTS ...
```

If any <triple bit> extends beyond the boundary of an <and spec> or an <or spec>, it will be truncated.

Appendix II: The Format of PLA Sources

In addition to the .COD files, PLASYS also supports a higher level specification language for PLAs. This is PLA source language is simply called the .SRC File Format. The PLASYS program ENCODE compiles .SRC files into .COD files. These .COD files are in the PLASYS .COD File Output Format.

[.COD files are the standard representation for PLAs. The formats for .COD files are described in Appendix I. All PLASYS programs accept and/or generate .COD files. Other programs (e.g. ICL or SIMULA PLA implementation systems) also accept and/or generate .COD files. The intention is that all Caltech software dealing with PLAs should use this format. Readers are directed to SSP File #3523 -- Proposed .COD File Formats by Dave Johannsen for more information.]

PLA source files start with a TITLE section and end with FINISH. In between there are three sections: ABBREVIATIONS, DECLARATIONS, and EQUATIONS. The following example shows a small PLA which implements the exclusive-or function.

```
TITLE "Exclusive OR PLA"
ABBREVIATIONS   xop SHORT FOR "AND NOT";
DECLARATIONS    INPUTS A B;     OUTPUTS AxorB;
EQUATIONS
        SET AxorB = (A /xop/ B) + (B /xop/ A);
FINISH
```

The remanider of this appendix is divided into four sections. They are:

A. Abbreviations
B. Declarations
C. Equations
D. Special Features

The first three refer to the three sections of a PLA source file. The last covers special features in the compiler to handle comments and collections of signals which are interpretted as a single numeric field.

A. Abbreviations

Abbreviations provide a simple text substitution facility within the ENCODE compiler. The ABBREVIATIONS section allows identifiers to be declared "short for" some text string. Wherever that identifier (enclosed in /'s) appears, ENCODE substitutes the appropriate text string. In the following example both equations are identical.

```
TITLE "Complex Use of Abbreviations"
ABBREVIATIONS   modifier SHORT FOR "(Alpha+Beta)";
                1st SHORT FOR "1 AND /modifier/";
```

```
                    2nd SHORT FOR "2 AND /modifier/";
DECLARATIONS        INPUTS X1 X2 Alpha Beta; OUTPUTS Z;
EQUATIONS
SET Z = X/1st/+X/2nd/;
SET Z = X1 AND (Alpha+Beta)+X2 AND (Alpha+Beta);
FINISH
```

B. Declarations

ENCODE supports three types of declarations. These are input signals, output signals, and feedbacks. Each source file may have several INPUTS and/or OUTPUTS declarations. They are effectively concatenated. Each signal appears in the PLA (.COD file) in the order it was declared. Thus, the input and output declarations can have implications on the actual PLA implementation.

The feedback declaration merely declares the number of feedback signals. The feedback signals automatically are given the names $0 through $n; n is one less than the number of feedbacks declared.

The first signal in the and-plane (inputs) is the first input declared. Inputs continue until all explicitly declared inputs have been assigned. Then the feedbacks are assigned starting with with $n and progressing down to $0 -- the last input to the and-plane. The total number of input columns is the sum of the declared INPUTS and the feedback count.

The first signal in the or-plane (outputs) is the first feedback ($0). The feedbacks then progress up to $n. The feedbacks are followed by the first explicitly declared output signal. The total number of output columns is the sum of the declared OUTPUTS and the feedback count.

If there is no feedback declaration, then the PLA will not have any feedback terms. In the following example the inputs are A,B,C,$1,$0; the outputs are $0,$1,X,Y,Z.

```
TITLE "Declaration Example"
DECLARATIONS
        OUTPUTS X Y; INPUTS A; FEEDBACK COUNT = 2;
        INPUTS B C; OUTPUTS Z;
EQUATIONS
        SET Z = B.C; SET X = B.A; SET Y = B C + B A;
        SET $0 = A $0 + A' B C; SET $1 = C $1 + C' A B;
FINISH
```

C. Equations

The equation section specifies the function of the PLA. Equations are comprised of two parts. These are called right-hand sides ⟨RHS⟩s and left-hand sides ⟨LHS⟩s. The ⟨LHS⟩s are the output signals which receive the value of the ⟨RHS⟩s. The ⟨RHS⟩s are expressions composed of input signals and operators.

There are two acceptable forms for equations. They are:

```
SET ⟨LHS⟩ = ⟨RHS⟩ ;
[ ⟨RHS⟩ ) ⟨LHS⟩ ]
```

The only difference between the these two forms is syntax.

The following two sections present some specifics related to equations. Particularly the modification of signals to describe complemented and don't care conditions, and the syntax for boolean expressions used in the specification of ⟨RHS⟩s.

1. Signal Modifiers

The output and feedback signal names in the ⟨LHS⟩s may be modified as complemented and don't care. A complemented output signal is not dependent on the ⟨RHS⟩ value. Effectively the signal is not listed in the ⟨LHS⟩. The value of a don't care output is "don't care" when the ⟨RHS⟩ is true. Most implementation will ignore these don't cares (default don't cares to false); some optimization routines take advantage of this additional flexibility.

The input and feedback signals names in the ⟨RHS⟩s may be similarly modified. A complemented input has the normal meaning. A don't care input is ignored. The following example shows the various cases of signal modification.

```
TITLE "Signal Modifiers"
DECLARATIONS
        OUTPUTS A B C;  INPUTS X Y Z;
EQUATIONS
        SET A = X Y' Z?;          %same as%       SET A = X and Y';
        SET B' = X;               %this equation is ignored%
        SET C? = X * ~Y;          %when xy' output c is don't care%
FINISH
```

2. ⟨RHS⟩ Expressions

The ⟨RHS⟩s are expressions are comprised of feedbacks and input signals combined with boolean operators. As described above, the basic signals can be modified -- complemented. Beyond this, they may be combined together into products. The forms of the product operator are AND, *, ., and &. Additionally, two signals with no operator between them are

also part of the same product (e.g. A B is the same as A & B). Products may be summed together; the sum operator forms are OR, +, and |. Additonally, parentheses may be used to form more complex boolean expressions. In the following example all three equations are the same.

```
TITLE "Boolean Expressions"
DECLARATIONS
        INPUTS I J K;   OUTPUTS X Y Z;
EQUATIONS
        SET X = (I+J)(I'+K);
        [ I I' | I K | J I' | J K > Y ]
        SET Z = (I and (not i or k)) + J & ~i + J . k;
FINISH
```

D. Special Features

Comments - Anywhere within the PLA source file any arbitrary text may be inserted provided it is enclosed in %'s. In addition, everything beyond the FINISH is ignored. The following example shows some of the possibilities.

```
TITLE "Example of Comments" %this is a comment% ;
ABBREVIATION P SHORT FOR "PU% "comments" can be anywhere %I";
DECLARATION     IN/P/ A B; % This comment is in the
                                declaration section%
                OUT/P/ X;
EQUATION        SET % Short Comment% X = A B;
FINISH  Everything after the FINISH is ignored.
```

Plurals - The plural keywords (DECLARATIONS, INPUTS, EQUATIONS etc.) are also recognized in their singular form. The previous example shows this.

Numeric Fields - The # operator is used to declare and use numeric fields. The syntax used with the # is a number comprised of 0's, 1's, and X's immediately followed by a #. This may be optionally followed by single letter. The expansion of a numeric is dependent on its context. the three contexts are declarations, <LHS>s, and <RHS>s. The following brief examples shows the contexts and expansions for 110xx#Q:

CONTEXT	EXPANSION
Declaration	Q4 Q3
<LHS> - output	Q4 Q3 Q1? Q0?
<RHS> - input	Q4 Q3 Q2'

In the case where no letter is specified the $ is used to reference the feedback terms. The following example shows the use of numerics. The example is incomplete; the full version of this

PLA is presented in Appendix III: An Example PLA Design using PLASYS.

```
TITLE "Example of Numerics"
ABBREVIATIONS is SHORT FOR " SHORT FOR ";
        STATE-EMPTY/is/" 0xx# ";
        STATE-ZERO /is/" 100# ";        STATE-ONE  /is/" 101# ";
        STATE-TWO  /is/ " 110# ";       STATE-THREE/is/" 111# ";
                ...
DECLARATIONS    FEEDBACK COUNT = 3;
        INPUTS  11#I;   % 2-bit input field %
                ...
EQUATIONS
        [ /ACTION-RECV/ 00#I     ) DATA-RECEIVED /STATE-ZERO/ ]
        [ /ACTION-RECV/ 01#I     ) DATA-RECEIVED /STATE-ONE/ ]
                ...
        [ /ACTION-XMIT/ /STATE-ZERO/     ) 00#O DATA-SENT ]
        [ /ACTION-XMIT/ /STATE-ONE/      ) 01#O DATA-SENT ]
                ...
FINISH
```

The PLASYS .SRC File Format

Syntax of .SRC Files Accepted the the PLASYS Program ENCODE

Note: ()'s enclose names of non-terminals. ()'s enclose sub-expressions. "'s enclose literal strings. /'s enclose descriptions not further expanded (e.g. /decimal integer/). []'s enclose optional parts of the syntax (e.g. [(a)] is the same as (a) ! /nothing/. {xyz}(n) means that xyz must appear exactly n times. If the (n) field is not present xyz must appear one or more times.

```
PLA SOURCE       := [ (title) ] [ (abbreviations) ]
                    [ (declarations) ] (equations) "FINISH"
(title)          := [ "TITLE" ] (quoted string) [ ";" ]
(abbreviations)  := "ABBREVIATIONS" [ ";" ] { (abbreviation) [ ";" ]}
 (abbreviation)  := (id) [ "SHORT" ] [ "FOR" ] (quoted string)
(declarations)   := "DECLARATIONS" [ ";" ] { (declaration) ";" }
(declaration)    := (input) ! (output) ! (feedback)
 (input)         := "INPUTS" { (id) }
 (output)        := "OUTPUTS" { (id) }
 (feedback)      := "FEEDBACK" [ "COUNT" ] [ "=" ] /number/
(equations)      := "EQUATIONS" [ ";" ] { (equation) [ ";" ] }
(equation)       := "SET" (lhs) "=" (rhs) ";" !
                    "[" (rhs) ")" (lhs) "]"
 (lhs)           := { ( (literal) ! (numeric) ) }
 (rhs)           := (product) ! (rhs) ( "+" ! "|" ! "OR" ) (product)
  (product)      := (primary) ! (product) [ (and) ] (primary)
   (and)                 := "&" ! "," ! "*" ! "AND"
  (primary)      := "(" (rhs) ")" ! { ( (literal) ! (numeric) ) }
  (numeric)      := /0,1,x number/ "$" [ /single letter/ ]
  (literal)      := (id) ! (prefix) (literal) ! (literal) (postfix)
   (prefix)      := "NOT" ! "~" (complement)
   (postfix)     := "'" ! "`" (complements) ! "?"  (don't care)
(id)             := /string of letters, digits, "_", "-", "!" or "$"/
(quoted string)  := """ /any text except "'s/ """
```

Appendix IIIa: An Example PLA Design

This example implements a state machine (M); M receives and transmits 2-bit data values. The reception protocol calls for M to receive (read) from a 2-bit input port (I0, I1) when DATA-PRESENTED goes high. When the port has been read, M acknowledges this by raising DATA-RECEIVED. The transmission protocol is similar. When DATA-REQUESTED is high, M can transmit (write) to a 2-bit output port (O0, O1). When the port is written, M reports this by raising DATA-SENT. M can receive and transmit in the same cycle.

This example makes two interesting uses of abbreviations. First, the state assignments are collected to a single place in the PLA source by the use of abbreviations (STATE-xxxx). This allows the state assignments to be easily changed. It is sometimes useful to experiment with different state assignments. This example uses the obvious assignment to good advantage.

The second use of abbreviation is to put some structure on the specification. It is clear that the BUFFER-AVAILABLE condition is true when the buffer is now empty (BUFFER-EMPTY) or being emptied (ACTION-XMIT). The more complex condition defined by the expansion of BUFFER-EMPTY and ACTION-XMIT is not so obviously correct. This use of nested abbreviations increases readability of designs and, hopefully, their ultimate correctness.

The implementation of M can be seen in the EQUATIONS section. Four actions have been defined in the ABBREVIATIONS section. These actions are: 1. Receive Data, 2. Transmit Data, 3. Refresh the Data (previously received which can not be transmitted yet), and 4. Reset.

1. Receive Data - There are four mutually exclusive equations which implement data reception. When ACTION-RECV is true, each possible input port value (00#I - 11#I) sets a different internal state. This comprise the first 4 equations.

2. Transmit Data - This is similar to data reception. When ACTION-XMIT is true, each of the internal states sets a different value into the output port. A code translation could be easily inserted here. This covers the next 4 equations.

3. Refresh Data - If nothing is happening the current internal state needs to be maintained. Here the next 3 equations refresh the internal state during ACTION-REFRESH.

4. Reset - Finally, when M has transmitted its data and not received any new data, it needs to reset back to the null state. Note that if a reset line was added to M, two changes would be necessary. First the INPUT declaration, and second the abbreviation for ACTION-RESET would be expanded.

This is the Source for the Example PLA:

```
TITLE "Example"
ABBREVIATIONS
        sf SHORT FOR " SHORT FOR ";     % this saves some typing %
        STATE-EMPTY/sf/" 0xx# ";
        STATE-ZERO /sf/" 100# ";        STATE-ONE  /sf/" 101# ";
        STATE-TWO  /sf/ " 110# ";       STATE-THREE/sf/" 111# ";
        ACTION-RECV /sf/ "(data-presented /buffer-available/)";
        ACTION-XMIT /sf/ "(data-requested /data-in-buffer/)";
        ACTION-RESET /sf/ "(/action-xmit/ not /action-recv/)";
        ACTION-REFRESH /sf/ "(/data-in-buffer/ not /action-xmit/)"
        DATA-IN-BUFFER /sf/ "(not/buffer-empty/)";
        BUFFER-EMPTY /sf/ "(/state-empty/)";
        BUFFER-AVAILABLE /sf/ "(/buffer-empty/ + /action-xmit/)";
DECLARATIONS
        FEEDBACK COUNT = 3;
        INPUTS  11#I;           % 2-bit input field %
        INPUT   data-presented; % external data in input field %
        OUTPUT  data-received;  % input field received %
        OUTPUTS 11#O;           % 2-bit output field %
        INPUT   data-requested; % external request for data %
        OUTPUT  data-sent;      % data sent to output field %
EQUATIONS
        [ /ACTION-RECV/ 00#I    ) DATA-RECEIVED /STATE-ZERO/ ]
        [ /ACTION-RECV/ 01#I    ) DATA-RECEIVED /STATE-ONE/ ]
        [ /ACTION-RECV/ 10#I    ) DATA-RECEIVED /STATE-TWO/ ]
        [ /ACTION-RECV/ 11#I    ) DATA-RECEIVED /STATE-THREE/ ]
        [ /ACTION-XMIT/ /STATE-ZERO/    ) 00#O DATA-SENT ]
        [ /ACTION-XMIT/ /STATE-ONE/     ) 01#O DATA-SENT ]
        [ /ACTION-XMIT/ /STATE-TWO/     ) 10#O DATA-SENT ]
        [ /ACTION-XMIT/ /STATE-THREE/   ) 11#O DATA-SENT ]
        [ /ACTION-REFRESH/ $0           ) $0 ]
        [ /ACTION-REFRESH/ $1           ) $1 ]
        [ /ACTION-REFRESH/ $2           ) $2 ]
        [ /ACTION-RESET/                ) /STATE-EMPTY/ ]
FINISH
```

ENCODE will compile this source into a PLA file. When this newly created PLA is printed with PPRINT the following PLA can be seen. The equations follow the PLA block printout.

```
!-------!-------!
!IIDD$$$!$$$DOOD!
!10AA210!012A10A!
!   TT  !   T  T!
!   AA  !   A  A!
!   --  !   -  -!
!   PR  !   R  S!
!   RE  !   E  E!
!   EQ  !   C  N!
!   SU  !   E  T!
!   EE  !   I   !
!   NS  !   V   !
!   TT  !   E   !
!   EE  !   D   !
!   DD  !       !
!-------!-------!
!     ? !  1    !
!     ? 1!1     !
!     ?1 ! 1    !
!    01  !  1   !
!    01 1!1     !
!    011 ! 1    !
!    1100!      1!
!    1101!     11!
!    1110!    1 1!
!    1111!    111!
!    1?  !..    !
!    ?1  !..    !
!   011  !..    !
!001 0   !  11  !
!00111   !  11  !
!011 0   !1 11  !
!01111   !1 11  !
!101 0   ! 111  !
!10111   ! 111  !
!111 0   !1111  !
!11111   !1111  !
!-------!-------!
```

Equations for: EXAMPLE

```
$0 = $2?.$0 + DATA-REQUESTED'.$2.$0 + [DATA-REQUESTED.$2?] +
     [DATA-REQUESTED?.$2] + [DATA-PRESENTED'.DATA-REQUESTED.$2] +
     I1'.I0.DATA-PRESENTED.$2' + I1'.I0.DATA-PRESENTED.DATA-REQUESTED.$2
      + I1.I0.DATA-PRESENTED.$2' + I1.I0.DATA-PRESENTED.DATA-REQUESTED.
     $2

$1 = $2?.$1 + DATA-REQUESTED'.$2.$1 + [DATA-REQUESTED.$2?] +
     [DATA-REQUESTED?.$2] + [DATA-PRESENTED'.DATA-REQUESTED.$2] +
     I1.I0'.DATA-PRESENTED.$2' + I1.I0'.DATA-PRESENTED.DATA-REQUESTED.$2
      + I1.I0.DATA-PRESENTED.$2' + I1.I0.DATA-PRESENTED.DATA-REQUESTED.
     $2

$2 = $2? + DATA-REQUESTED'.$2 + I1'.I0'.DATA-PRESENTED.$2' +
     I1'.I0'.DATA-PRESENTED.DATA-REQUESTED.$2 + I1'.I0.DATA-PRESENTED.$2
     ' + I1'.I0.DATA-PRESENTED.DATA-REQUESTED.$2 + I1.I0'.DATA-PRESENTED
     .$2' + I1.I0'.DATA-PRESENTED.DATA-REQUESTED.$2 + I1.I0.
     DATA-PRESENTED.$2' + I1.I0.DATA-PRESENTED.DATA-REQUESTED.$2

DATA-RECEIVED = I1'.I0'.DATA-PRESENTED.$2' + I1'.I0'.DATA-PRESENTED.
     DATA-REQUESTED.$2 + I1'.I0.DATA-PRESENTED.$2' + I1'.I0.
     DATA-PRESENTED.DATA-REQUESTED.$2 + I1.I0'.DATA-PRESENTED.$2' +
     I1.I0'.DATA-PRESENTED.DATA-REQUESTED.$2 + I1.I0.DATA-PRESENTED.$2'
      + I1.I0.DATA-PRESENTED.DATA-REQUESTED.$2

O1 = DATA-REQUESTED.$2.$1.$0' + DATA-REQUESTED.$2.$1.$0

O0 = DATA-REQUESTED.$2.$1'.$0 + DATA-REQUESTED.$2.$1.$0

DATA-SENT = DATA-REQUESTED.$2.$1'.$0' + DATA-REQUESTED.$2.$1'.$0 +
     DATA-REQUESTED.$2.$1.$0' + DATA-REQUESTED.$2.$1.$0
```

By running SHRINK and PAIRS this PLA can be reduced to 4/7 its original size. The final PLA is:

```
!-------!-------!
!IIDD$$$!$$$DOOD!
!10AA210!012A10A!
!  TT   !   T  T!
!  AA   !   A  A!
!  --   !   -  -!
!  PR   !   R  S!
!  RE   !   E  E!
!  EQ   !   C  N!
!  SU   !   E  T!
!  EE   !   I   !
!  NS   !   V   !
!  TT   !   E   !
!  EE   !   D   !
!  DD   !       !
!-------!-------!
!   0i  !  1    !
!   0i 1!1      !
!   0ii ! 1     !
!   ii  !      1!
!   ii 1!     1 !
!   iii !    1  !
!  1 0  !  11   !
!  1i1  !  11   !
! 11 0  !1       !
! 1111  !1       !
!1 1 0  ! 1     !
!1 1i1  ! 1     !
!-------!-------!
```

Equations for: EXAMPLE

```
$0 = DATA-REQUESTED',$2,$0 + I0.DATA-PRESENTED.$2' +
        I0.DATA-PRESENTED.DATA-REQUESTED.$2
$1 = DATA-REQUESTED',$2,$1 + I1.DATA-PRESENTED.$2' +
        I1.DATA-PRESENTED.DATA-REQUESTED.$2
$2 = DATA-REQUESTED',$2 + DATA-PRESENTED.$2' +
        DATA-PRESENTED.DATA-REQUESTED.$2
DATA-RECEIVED = DATA-PRESENTED.$2' + DATA-PRESENTED.DATA-REQUESTED.$2
O1 = DATA-REQUESTED.$2.$1
O0 = DATA-REQUESTED.$2.$0
DATA-SENT = DATA-REQUESTED.$2
```

Finally, this is the PLA (.COD) File for the final PLA designed using PLASYS.

```
I=4 F=3 O=4 M=12 EXAMPLE OF NUMERICS AND ABBREVIATIONS
...01.. 0010000
...01.1 1000000
...011. 0100000
...11.. 0000001
...11.1 0000010
...111. 0000100
..1.0.. 0011000
..111.. 0011000
.11.0.. 1000000
.1111.. 1000000
1.1.0.. 0100000
1.111.. 0100000
I1
I0
DATA-PRESENTED
DATA-REQUESTED
DATA-RECEIVED
O1
O0
DATA-SENT

     PLA Area:   441
SHRINK: Collapse unnecessary parts of the PLA Specification [6.2]
Number of unused terms deleted =  3.
Number of error terms deleted  =  3.
     PLA Area:   315
PAIRS: Combine compatible pairs of terms [6.1]
Total Pair-wise Combines =  1.
Total Simplifications    =  7.
Net Term Reduction =   1.
     PLA Area:   294
PAIRS: Combine compatible pairs of terms [6.1]
Total Pair-wise Combines =  2.
Total Simplifications    =  2.
Net Term Reduction =   2.
     PLA Area:   252
```

The PLASYS programs used for this example were: ENCODE, SHRINK, PAIRS, and PPRINT.

Appendix IIIb: Mead and Conway PLA Example

The following example in documented in "Introduction to VLSI Systems" by Mead and Conway [Mea80]. The example is a traffic light controller. The following PLA Source File is modeled closely to Table 3.1 on page 86 of the book:

```
TITLE "Mead & Conway Traffic Light Example"
ABBREVIATIONS
        % coding values for traffic light outputs %
        GRN short for " 00#";    % set light GREEN %
        YLW short for " 10#";    % set light YELLOW %
        RED short for " 01#";    % set light RED %
        % state assignments for finite state machine %
        HG short for " 00# ";    % Highway Green %
        HY short for " 10# ";    % Highway Yellow %
        FY short for " 01# ";    % Farmroad Yellow %
        FG short for " 11# ";    % Farmroad Green %
DECLARATIONS
        FEEDBACK COUNT = 2;      % 2 State Variables - 4 States %
        INPUT   C;               % Cars waiting on the Farmroad %
        INPUT   TS TL;           % Short and Long Timeout Indications %
        OUTPUTS 11#F;            % Farmroad Light Control Outputs %
        OUTPUTS 11#H;            % Highway Light Control Outputs %
        OUTPUT ST;               % Start Timeout Timer Output Command %
EQUATIONS
        % Table 3.1 - Page 86 in Mead and Conway %
        %Current ---Inputs---      Next  ---Outputs---   %
        % State                    State                 %
        [ /HG/  NOT (C and TL)   ) /HG/  /GRN/H /RED/F ]
        [ /HG/  (C and TL)       ) /HY/  /GRN/H /RED/F ST ]
        [ /HY/  NOT TS           ) /HY/  /YLW/H /RED/F ]
        [ /HY/  TS               ) /FG/  /YLW/H /RED/F ST ]
        [ /FG/  NOT (C' or TL)   ) /FG/  /RED/H /GRN/F ]
        [ /FG/  (C' or TL)       ) /FY/  /RED/H /GRN/F ST ]
        [ /FY/  NOT TS           ) /FY/  /RED/H /YLW/F ]
        [ /FY/  TS               ) /HG/  /RED/H /YLW/F ST ]
FINISH
```

This is a listing of the optimized version of the example PLA. Both QUINE and PAIRS found improvements to the original encoding. This listing shows the PLA and the simplified equations.

```
   <<<MEAD & CONWAY TRAFFIC LIGHT EXAMPLE>>>
     PLA Area:   170
QUINE: Minimize each equation
  Number of product terms uses deleted =  4.
  Number of PLA rows deleted =  1.
     PLA Area:   153
PAIRS: Combine compatible pairs of terms [6.1]
Total Simplifications   =   1.
     PLA Area:   153
.COD File Pretty Printer - version 6.0 (March 11, 1980)

          This printout was generated 1980-05-06 at 21:21:36.

 Inputs    =    3 Outputs   =    5
 Feedbacks =    2 Minterms  =    9
     PLA Area:   153

!-----!-------!
!CTT$$!$$FFHHS!
! SL10!011010T!
!-----!-------!
!     0!    1    !
!    10! 1   1   !
!   111!1     11!
! 0 01!1 1   1  !
! 1 01!   1   11!
! 1 10!1       1!
!0   11!1     11!
!1 011!11    1  !
!1 100! 1     1!
!-----!-------!
```

Equations for: MEAD & CONWAY TRAFFIC LIGHT EXAMPLE

```
$0 = TL.$1.$0 + TS'.$1'.$0 + TS.$1.$0' + C'.$1.$0 + C.TL'.$1.$0
$1 = $1.$0' + C.TL'.$1.$0 + C.TL.$1'.$0'
F1 = TS'.$1'.$0 + TS.$1'.$0
F0 = $0'
H1 = $1.$0'
H0 = TL.$1.$0 + TS'.$1'.$0 + TS.$1'.$0 + C'.$1.$0 + C.TL'.$1.$0
ST = TL.$1.$0 + TS.$1'.$0 + TS.$1.$0' + C'.$1.$0 + C.TL.$1'.$0'
```

This is the final PLA .COD File:

```
I=3 F=2 O=5 M=9 MEAD & CONWAY TRAFFIC LIGHT EXAMPLE
....0 0001000
...10 0100100
..111 1000011
.0.01 1010010
.1.01 0010011
.1.10 1000001
0..11 1000011
1.011 1100010
1.100 0100001
C
TS
TL
F1
F0
H1
H0
ST

     PLA Area:   170
QUINE: Minimize each equation
  Number of product terms uses deleted =  4.
  Number of PLA rows deleted =  1.
     PLA Area:   153
PAIRS: Combine compatible pairs of terms [6.1]
Total Simplifications   =   1.
     PLA Area:   153
```

Appendix IV: PLASYS Implementation and Maintenance

PLASYS is a series of Simula programs; collectively they are called the PLASYS application programs. Most applications read PLA descriptions from .COD files (see Appendix I), perform some operation (optimization?) on the PLA, and write a new .COD file of the same name. One application (ENCODE) reads PLA source files (see Appendix II) and generates .COD files. Another (PPRINT) reads the standard PLA description (.COD) files and generates documentation (human readable) listing (.LST) files.

This appendix describes the basic details of the implementation. Information required by PLASYS users has been included in the main document and the preceding appendices. This appendix is provided as system documentation; the listings have only minimal comments.

This appendix is divided into four sections. They are:

A. The PLA Data Structure
B. General Structure of a PLASYS Application
C. The PLASYS Application Skeleton
D. The files on PS:<SSPLIB.PLA>

A. The PLA Data Structure

The basic data structure element is a class which is an array of texts. The class is used to support dynamic allocation and deallocation of these arrays. In addition to the array, the class also supports a size field. An internal real_size field is used by the routines which add and delete elements from the arrays.

The application (in-core) data structure for a PLA is 4 arrays of texts, 7 integers, a regular Simula text array, and a text. The text HEADER_LINE contains the name of the PLA; the PLA's name is the remainder of the .COD file header line - if this is blank in the .COD file, the read routines create a name from the .COD file name and today's date. The text array COMMENTS contains the first block of comments from the .COD file. This array has a fixed allocation of 1:99. Any comments beyond this count are thrown away.

The 7 integers are:

I_EQUALS	The number of inputs (I)
O_EQUALS	The number of outputs (O)
M_EQUALS	The number of minterms (M)
F_EQUALS	The number of feedbacks (F)
C_EQUALS	The number of comments (0 <= C <= 99)
IF_EQUALS	The total number of input columns (I+F)
OF_EQUALS	The total number of output columns (O+F)

The 4 dynamic text arrays are:

1. OUTPUT_NAMES - This array has O+F entries. The first F entries are the feedback names ($0 - $F). This is followed by the names of the other output signals.

2. INPUT_NAMES - This array has I+F entries. The last F entries are the names of the feedbacks ($F - $0). The remainder of the array has the names of the input signals.

3. AND_ROWS - This array has M entries. Each entry is a text of I+F characters from the set (0,1,.,?). Each entry represents a product term (minterm) in the PLA. The ith character in the array represents the ith input column in the PLA; its name is the ith entry in INPUT-NAMES.

   The interpretation of the characters for an example input signal (column) S in entry M of AND_ROWS is as follows:

0	When S' is FALSE, minterm M is FALSE.
1	When S is FALSE, minterm M is FALSE.
.	Minterm M does not depend on S.
?	When S.S' is FALSE, minterm M is FALSE.

   (PPRINT suppresses the printing of .'s)

4. OR_ROWS - This array has M entries. Each entry is a text of O+F characters from the set (0,1,.,?). Each entry represents all uses of the corresponding product term in AND_ROWS. The ith character in the array represents the ith output column in the PLA; its name is the ith entry in OUTPUT_NAMES.

   The interpretation of the characters for an example output signal (column) X in entry M of OR_ROWS is as follows:

0	X does not depend on minterm M.
1	When minterm M is TRUE, X is TRUE.
.	When minterm M is TRUE, X is "don't care"
?	Error

   (PPRINT suppresses the printing of 0's)

B. General Structure of a PLASYS Application

Most PLASYS Application programs have four files associated with them. These files all have the same file name. As CHECK is the simplest application it will be used as the example. These four files are combined with common files to form each application. The four files are:

1. Application Source - CHECK.SIM
2. Application Messages - CHECK.MSG

3. Application Command File - CHECK.CMD
4. Application Executable File - CHECK.EXE

1. Application Source

The CHECK source is in file CHECK.SIM. The following is a complete copy of CHECK.SIM:

```
COMMENT          CHECK.SIM;
TEXT PROCEDURE name_of_application;
name_of_application :- Copy("CHECK");

PROCEDURE application;
BEGIN   END;
```

Application source programs contain the procedures which need to be added to the application skeleton. The skeleton reads and writes .COD files, interacts with the user, builds log files etc. Thus, CHECK has very few responsibilites.

The skeleton calls two procedures. The first is the text procedure name_of_application. The text value is the application's name. This name is used to find the message file (see 2. below), and as the log file's file name. This procedure is called before the skeleton does anything interesting. ENCODE sets a flag in name_of_application which suppresses the reading of the .COD file. (ENCODE needs to compile a .SRC file to create a .COD - not read an existing .COD files as with all other applications.)

The second procedure is application; this procedure does everything that needs to be done between reading and writing the .COD file. In the case of CHECK, nothing needs to be done.

2. Application Messages

Most messages printed on the user's terminal come from message files. These files have the extension .MSG. The following is a complete copy of CHECK.MSG:

```
.COD File Checker - version 6.0 (March 11, 1980)
REQ
*       .COD File Checker - version 6.0 (March 11, 1980)*
**              CHECK reads a .COD file and outputs one which follows*
**              the PLASYS stricter syntax. Errors in the file are*
**              reported, but not necessarily fixed.*
?       xxxx.COD (= xxxx.COD - Please enter xxxx: ?
REQ
```

. Message files are accessed by the procedure message. Message takes two parameters - the message file name (extension .MSG assumed), and a keyword. The message procedure opens the file and searches for the first line containing the keyword. All subsequent lines are "processed" until the matching keyword is encountered. (If someone wished to make the program more terse, the .MSG files could be shortened. Each "processed" line prints all but the first and last character followed by a carriage return. If the first character is a ? the carraige return is suppressed. If this character is a +, the error count is incremented.

The skeleton accesses the application's message file by using the name_of_application procedure. Much of the user interface (what there is of it) has been isolated to the .MSG files.

3. Application Command File

The application's command file is used to compile, load, and save a new version of the application. This use done whenever the application source changes. When the skeleton is changed, all applications must be re-compiled. The command file for CHECK is CHECK.CMD. It contains the following:

```
COPY FIRST.SIM T.SIM
APPEND UTIL.SIM T.SIM
APPEND IO.SIM T.SIM
APPEND CHECK.SIM T.SIM
APPEND LAST.SIM T.SIM
Simula
LOAD T
SAVE CHECK.EXE
DELETE T.SIM, T.REL
```

. Sometime during the TAKE, Simula will require a file name. All command files expect the same response: T^Z ("T" - control-Z). The command file simply concatenates the skeleton sources with the application sources, and compiles a single large Simula program. This is loaded and saved. The temporary combined Simula source and relocatable files are deleted.

4. Application Executable File

The executable file has the standard extension .EXE. It is self-contained and may be executed anywhere, and copied or renamed at will, WITH ONE EXCEPTION... Whenever the application is executed the user must have direct access to the message files with their original names. The user must either have these files on their disk area, or accessible through their search list. In addition to the application message file, the user must also be able to access the skeleton's message files.

C. The PLASYS Application Skeleton

PLASYS applications are formed by concatenating the skeleton source files with source files specific to the application. The details of this process were presented in section B (General Structure of a PLASYS Application) above. This section presents the functions performed by the skeleton source files. The skeleton source files are:

1. FIRST.SIM
2. UTIL.SIM
3. IO.SIM
4. OPS.SIM
5. LAST.SIM

The skeleton also includes a message file (PLASYS.MSG) which contains messages used by the skeleton.

1. FIRST.SIM - FIRST declares the externals and classes needed by the PLASYS applications. It also declares the global variables used to store the PLA during application execution. The basic procedure has the skeleton loading these global variables, the applications manipulating the PLA in this global data space, and the skeleton writing a new version of the .COD file from these variables. All applications must start with FIRST.

2. UTIL.SIM - UTIL has a number of utility routines. These procedures open files, do all the log file functions, and the message function. All applications must include UTIL. Some other special functions in util include: a. TRIPLE - concatenate three TEXTs, b. WIDTH - printing width of a number (integer version of log base 10), c. RAISE - make lower case to upper, return all other inputs unchanged, and d. MAKE - make a TEXT length N of character C everywhere except put a character S at position P.

3. IO.SIM - IO does the PLA read and write functions. All applications must include IO. Additionally IO contains procedures to calcualte areas of a PLA and generate indexed names (e.g. $0, $1, $2 etc.).

4. OPS.SIM - OPS has various text string operators and class manipulation operators. Some applications do not use OPS. OPS has the procedures which add and delete elements from the dynamically size arrays of integers and texts which are used to represent the PLA in memory. OPS also contains the lookup procedure to locate a text in a dynamic text array.

   OPS also contains the operators. The basis of the operators takes advantage of the ASCII codes for the 4 characters used internally to represent PLAs. These characters and their codes in binary are: 0: 000000, 1: 000001, .: 101110, and ?: 111111. It just

so happens that these codes are all different in their lower-order two bits. Thus, operators are represented as two-dimensional character arrays 0:3 by 0:3. These operators are initialized with a string of 16 characters of the form "abcdefghijklmnop" with the following interpretation:

```
Operand 1:      0000111....????
Operand 2:      01.?01.?01.?01.?
Result:         abcdefghijklmnop
```

. Thus a reasonable AND function would be initialized with "000001.?0..?0???".

5. LAST.SIM - LAST contains the main program.

D. The Files on PS:<SSPLIB.PLA>

The disk area PS:<SSPLIB.PLA> contains about 100 files which use about 450 disk pages.

```
AAxxx.DOC:  These are the primary text files for this document.
AAxxx.FMT:  These are the paged/formatted copies of this document.
Axxxx.DEC:  .COD file for a Digital Equipment PLA.
Axxxx.XRX:  .COD file for a Xerox PLA.
CHECK.xxx:  Files for CHECK Application (see B. above).
DELPLASYS:  A user command file to delete .LOG and .MSG files.
ENCODE.xxx: Files for the ENCODE Application (see B. above).
FIRST.SIM:  A source file for the PLASYS skeleton (see C. above).
GETPLASYS:  A user command file to copy .MSG files from <SSPLIB.PLA>.
IO.SIM:     A source file for the PLASYS skeleton (see C. above).
LAST.SIM:   A source file for the PLASYS skeleton (see C. above).
LIGHTS.xxx: PLASYS example from Mead and Conway (Appendix III)
MUTEX.xxx:  Files for the MUTEX Application (see B. above).
OPS.SIM:    A source file for the PLASYS skeleton (see C. above).
OPTIMIZE:   A user command file to optimize a .COD file.
PAIRS.xxx:  Files for the PAIRS Application (see B. above).
PLASYS.MSG: Message file for PLASYS skeleton (see C. above).
PPRINT.xxx: Files for the PPRINT Application (see B. above).
QUINE.xxx:  Files for the QUINE Application (see B. above).
REDUCE.xxx: Files for the REDUCE Application (see B. above).
RHS.SIM:    Additional source file for ENCODE Application.
ROTATE.xxx: Complete ROTATE Program.
SAMPLE.xxx: PLASYS example (see Appendix III above).
SHRINK.xxx: Files for the SHRINK Application (see B. above).
SPREAD.xxx: Files for the SPREAD Application (see B. above).
TEST.xxx:   Source for PLA to test PLASYS.
XQUINE.xxx: Files for the XQUINE Application (see B. above).
```

Appendix V: PLA Optimization Bibliography

Are78: Arevalo, Zosimo, and Jon G. Bredeson, "A Method to Simplify a Boolean Function into a Near Minimal Sum-of-Products for Programmable Arrays," IEEE Transactions on Computers, Vol. C-27, No. 11, pp. 1028-1039, November 1978.

Ayr79: Ayres, Ron, "Silicon Compilation - A Hierarchical Use of PLAs," in Proceeding of 16th ACM Design Automation Conference, pp. 314-326, June 1979.

Bar61: Bartee, Thomas C., "Computer Design of Multiple-Output Logical Networks," IRE Transactions on Electronic Computers, Vol. 10, pp. 21-30, March 1961.

Bow70: Bowman, Robert M., and E. S. McVey, "A Method for the Fast Approximate Solution of Large Prime Implicant Charts," IEEE Transactions on Computers, Vol. C-19, No. 2, pp. 169-173, February 1970.

Bre72: Breuer, Melvin A., "Logic Synthesis - Section 2.3 - Covering a Single-Output Switching Function" in Design Automation of Digital Systems, Vol. 1, pp. 33-49, Prentice-Hall Inc., Englewood Cliffs, N.J., 1972.

Bri99: Bricaud, Pierre, and John Campbell, "Multiple Output PLA Minimization: EMIN" ...

Cha78: Cha, Charles W., "A Testing Strategy for PLAs," in Proceedings of 15th ACM Design Automation Conference, Las Vegas, NV., pp. 326-334, June 19-21 1978.

Coo79a: Cook, Peter W., etal, "One Micron MOSFET PLAs", in Proceedings of 1979 IEEE International Solid-State Circuits Conference, February 1979.

Coo79b: Cook, Peter W., Ho, Chung W., and Stanley E. Schuster, "A Study in the Use of PLA-Based Macros," IEEE Journal of Solid-State Circuits, Vol. SC-14, No. 5, pp. 833-840, October 1979.

DeV75: De Vries, Ronald C., and Antonin Svoboda, "Multiple-Output Optimization with Mosaics of Boolean Functions," IEEE Transactions on Computers, Vol. C-24, No. 8, pp. 777-785, August 1975.

Dun59: Dunham, B., and R. Fridshal, "The Problem of Simplfying Logical Expressions," The Journal of Symbolic Logic, Voo Vol. 24, No. 1, pp. 17-19, March 1959.

Eic80: Eichelberger, E. B., and E. Lindbloom, "A Heuristic Test-Pattern Generator for Programmable Logic Arrays," IBM Journal of Research and Development, Vol. 24, No. 1, pp. 15-22, January 1980.

Fle75: Fleisher, H., and L. I. Maissel, "An Introduction to Array Logic," IBM Journal of Research and Development, Vol. 19, pp. 98-109, March 1975.

Fle79: Fleisher, H., "Introduction to Special Section on Programmable Logic Arrays," IEEE Transactions on Computers, Vol. C-28, No. 9, pp. 593, September 1979.

Gol80: Golden, R. L., Latus, P. A., and P. Lowry, "Design Automation and the Programmable Logic Array Macro," IBM Journal of Research and Development, Vol. 24, No. 1, pp. 23-31, January 1980.

Hil68: Hill, F.J., and G.R. Peterson, "Introduction to Switching Theory and Logical Design," John Wiley & Sons, Inc., New York, N.Y., 1968, pp. 122-140.

Hon74: Hong, S. J., R. G. Cain, and D. L. Ostapko, "MINI: A Heuristic Approach for Logic Minimization," IBM Journal of Research and Development, Vol. 18, pp. 443-458, September 1974.

Hul75: Hulme, Bernie L., and Richard B. Worrell, "A Prime Implicant Algorithm with Factoring," IEEE Transactions on Computers, Vol. C-24, No. 11, pp. 1129-1131, November 1975.

Jon75: Jones, J. W., "Array Logic Macros," IBM Journal of Research and Development, Vol. 19, pp. 120-126, March 1975.

Kam79: Kambayashi, Y., "Logic Design of Programmable Logic Arrays," IEEE Transactions on Computers, Vol. C-28, No. 9, pp. 609-617, September 1979.

Log75: Logue, J. C., etal, "Hardware Implementation of a Small System in Programmable Arrays," IBM Journal of Research and Development, Vol. 19, pp. 110-119, March 1975.

McC62: McCluskey, E. J. Jr., and H. Schorr, "Essential Multiple-Output Prime Implicants," Princeton University, Dept. of Electrical Engineering, Digital Systems Laboratory, Technical Report No. 23, April 1962.

Mea80: Mead, Carver, and Lynn Conway, "Introduction to VLSI Systems," Addison-Wesley Publishing Co., Reading, Mass., 1980, pp. 885-88. Mor70: Morreale, Eugenio, "Recursive Operators for Prime Implicant and Irredundant Normal Form Determination," IEEE Transactions on Computers, Vol. C-19, No. 6, pp. 504-509, June 1970.

Ost74: Ostapko, D. L., and S. J. Hong, "Generating Test Examples for Hueristic Boolean Minimization," IBM Journal of Research and Development, Vol. 18, pp. 459-464, September 1974.

Ost79: Ostapko, D. L., and S. J. Hong, "Fault Analysis and Test Generation for Programmable Logic Arrays PLA's:," IEEE Transactions on Computers, Vol. C-28, No. 9, pp. 617-627, September 1979.

Pat79: Patil, S. S., and T. A. Welch, "A Programmable Logic Approach for VLSI," IEEE Transactions on Computers, Vol. C-28, No. 9, pp. 594-601, September 1979.

Reu75: Reusch, Bernd, "Generation of Prime Implicants from Subfunctions and a Unifying Approach to the Covering Problem," IEEE Transactions on Computers, Vol. C-24, No. 9, pp. 924-930, September 1975.

Rhy77: Rhyne, V. Thomas, Philip S. Noe, Melvin H. McKinney, and Udo W. Pooch, "A New Technique for the Fast Minimization of Switching Functions," IEEE Transactions on Computers, Vol. C-26, No. 8, pp. 757-763, August 1977.

Sch80: Schmookler, "Design of Large ALUs Using Multiple PLA Macros," IBM Journal of Research and Development, Vol. 24, No. 1, pp. 2-14, January 1980.

Sla70: Slagle, James R., Chin-Liang Chang, and Richard C. T. Lee, "A New Algorithm for Generating Prime Implicants," IEEE Transactions on Computers, Vol. C-19, No. 4, pp. 304-310, April 1970.

Su69: Su, Yueh-Hsung, and Donald L. Dietmeyer, "Computer Reduction of Two-Level, Multiple-Output Switching Circuits," IEEE Transactions on Computers, Vol. C-18, No. 1, pp. 58-63, January 1969.

Su72: Su, Stephen Y. H., and Petter T. Cheung, "Computer Minimization of Multivalued Switching Functions," IEEE Transactions on Computers, Vol. C-21, No. 9, pp. 995-1003, September 1972.

Sur75: Sureschander, "Minimization of Switching Functions - A Fast Technique," IEEE Transactions on Computers, Vol. C-24, No. 7, pp. 753-756, July 1975.

Svo75: Svoboda, Antonin, "The Concept of Term Exclusiveness and Its Effect on the Theory of Boolean Functions," Journal of the ACM, Vol. 22, No. 3, pp. 425-440, July 1975.

Wan79: Wang, Shean Lin, and Algirdas Avizienis, "The Design of Totally Self Checking Circuits Using Programmable Logic Arrays," presented at International Symposium on Fault-Tolerant Computing, Madison, WI., June 20-22, 1979.

Wei79: Weinberger, Arnold, "High-Speed Programmable Logic Array Adders", IBM Journal of Research and Development, Vol. 23, pp. 163-178, March 1979.

Web79: Weber, Helmut, "High Level Design for Programmed Logic Arrays" in Proceedings of the 4th International Symposium on Computer Hardware Description Languages, Palo Alto, CA, pp. 96-101, October 1979.

Woo79: Wood, R. A., "A High Density Programmable Logic Array Chip," IEEE Transactions on Computers, Vol. C-28, No. 9, pp. 602-608, September 1979.